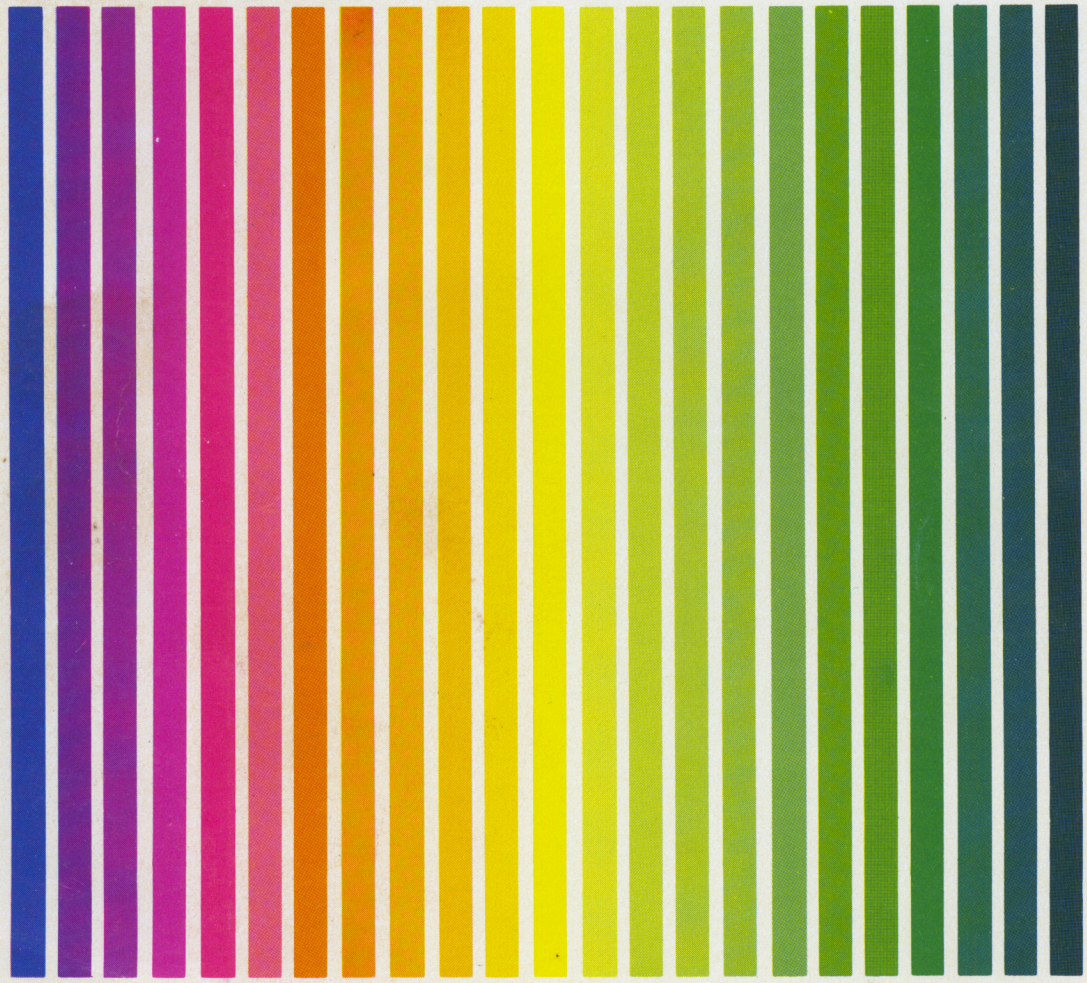
APX ATARI® PROGRAM EXCHANGE



The Soft Warehouse

POLYCALC™

Perform symbolic algebra and calculus
with polynomials (ages 14 and up)

Cassette: 24K (APX-10127) Diskette: 32K (APX-20127)

User-Written Software for ATARI Home Computers

The Soft Warehouse

POLYCALC™

Perform symbolic algebra and calculus
with polynomials (ages 14 and up)

Cassette: 24K (APX-10127) Diskette: 32K (APX-20127)

Trademark Notice

POLYCALC is a trademark of The Soft Warehouse

Disclaimer

The Soft Warehouse makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Further, The Soft Warehouse reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of The Soft Warehouse to notify any person or organization of such revision or changes.

POLYCALC

by

The Soft Warehouse

Program and Manual Contents © 1982 The Soft Warehouse

Copyright notice. On receipt of this computer program and associated documentation (the software), the author grants you a nonexclusive license to execute the enclosed software. This software is copyrighted. You are prohibited from reproducing, translating, or distributing this software in any unauthorized manner.

Distributed By

The ATARI Program Exchange
P.O. Box 3705
Santa Clara, CA 95055

To request an APX Product Catalog, write to the address above, or call toll-free:

800/538-1862 (outside California)
800/672-1850 (within California)

Or call our Sales number, 408/727-5603

Trademarks of Atari

The following are trademarks of Atari, Inc.

ATARI®
ATARI 400™ Home Computer
ATARI 800™ Home Computer
ATARI 410™ Program Recorder
ATARI 810™ Disk Drive
ATARI 820™ 40-Column Printer
ATARI 822™ Thermal Printer
ATARI 825™ 80-Column Printer
ATARI 830™ Acoustic Modem
ATARI 850™ Interface Module

Printed in U.S.A.

TABLE OF CONTENTS

INTRODUCTION . . . 4
Overview . . . 4
Required Accessories . . . 4
Contacting the author . . . 4

GETTING STARTED . . . 5
Loading POLYCALC into computer memory . . . 5

USING POLYCALC . . . 6
Entering problems . . . 6
Interrupting computation . . . 6
the kinds of operations POLYCALC can perform . . . 6

THE HELP TOPICS . . . 7
Expressions . . . 9
Variables . . . 9
Constants . . . 9
Scientific notation . . . 9
Signed numbers . . . 10
Order of operations . . . 10
Multiple exponentiation . . . 10
Multiple multiplication and division . . . 10
Multiple addition and subtraction . . . 10
Indicating division . . . 10
Integer exponents . . . 11

IMPROPERLY FORMED EXPRESSIONS . . . 11

UNSTORABLE EXPRESSIONS . . . 11
Accuracy and Rounding . . .13
Limitations on magnitudes of numbers . . . 14

ASSIGNMENTS . . . 14
Suppressed Display and Compound Entries . . . 18
Suppressed Evaluation . . . 18
Re-evaluation . . . 20

USING THE CALCULUS FEATURE . . . 21
Partial Derivatives . . . 21
Intergrals . . . 21
Truncated Power Series . . . 22

How To Learn More About Computer Algebra . . . 25
The Professional Societies . . . 25
The Literature . . . 25
Other computer algebra . . . 26

COMPUTER ALGEBRA IN EDUCATION . . . 28

ALPHABETIZED ERROR MESSAGES AND CAUSES . . . 30

QUICK REFERENCE SHEET . . . 32

INTRODUCTION

Overview

POLYCALC is a computational tool for performing symbolic algebra and calculus operations. This program supports polynomials that are generalized to permit fractional and negative powers of variables. The program permits up to 26 unassigned variables. POLYCALC is essentially a polynomial system rather than a rational expression system. The program can help professionals, students, and teachers of algebra or calculus with their computational activities, including integration.

POLYCALC has an advantage over other programming languages like BASIC, FORTRAN, and FORTH, which require that numeric values be assigned to all variables in expressions before evaluating the expression. In POLYCALC you don't have this limitation.

POLYCALC is easy to use, and it has help screens you can display while using the program. To perform an operation, you type in an expression, and the program then expands, integrates, or differentiates the expression, as you request. After the result displays, you can enter another expression. In addition, you can assign the results to variables for use in later expressions, letting you perform a sequence of related operations.

REQUIRED ACCESSORIES

ATARI BASIC Language Cartridge

Cassette version

24K RAM
ATARI 410 Program Recorder

Diskette version

32K RAM
ATARI 810 Disk Drive

Users wishing to contact the author about POLYCALC may write to The Soft Warehouse at:

P.O. Box 11174
Honolulu, Hawaii 96828

GETTING STARTED

LOADING POLYCALC INTO COMPUTER MEMORY

1. Insert the ATARI BASIC Language Cartridge in the cartridge slot of your computer.

2. If you have the cassette version of POLYCALC:

   a. Turn on your TV set.

   b. Connect your program recorder to the computer and to a wall outlet.

   c. Slide the POLYCALC cassette into the program recorder's cassette holder and press REWIND on the recorder until the tape rewinds completely. Then press PLAY.

   d. Turn on your computer while holding down the START key. Press the RETURN key. POLYCALC will start automatically.

If you have the diskette version of POLYCALC:

   a. Turn on your disk drive.

   b. When the BUSY light goes out, open the disk drive door and insert the POLYCALC diskette with the label in the lower right-hand corner nearest to you. Close the door.

   c. Turn on your computer and TV set. POLYCALC will load and run automatically.

THE FIRST DISPLAY SCREEN

The first program display contains the title and copyright information, together with a prompt for requesting lessons:

```
|POLYCALC (tm) Polynomial Program      |
|Atari  version                        |
|Copyright (c) 1981 The Soft Warehouse|
|Box 11174, Honolulu, Hawaii 96828     |
|                                      |
|Enter another question mark for help  |
|                                      |
|?_                                    |
|--------------------------------------|
```

After the title screen displays, you may either start entering problems immediately or select a lesson to practice using POLYCALC's features.

USING POLYCALC

ENTERING PROBLEMS

It's easy to use POLYCALC for solving problems. You type in expressions, using the conventions and notation described in the help topic and in this manual. If you make a typing mistake, use the DELETE/BACK S key to erase the error and type in the correction. Once you've entered the problem accurately, you press the RETURN key to signal the program to process your problem. POLYCALC beeps when it completes the operation and displays the results. This feature lets you turn your attention elsewhere when POLYCALC is doing time-consuming operations, since it notifies you audibly when you can enter another problem.

INTERRUPTING COMPUTATIONS

To interrupt a computation, press the START key. This action stops not only the current computation but also the program's functioning. At this point the program will ask you to enter another problem.

THE KINDS OF OPERATIONS POLYCALC CAN PERFORM

POLYCALC can perform symbolic algebra and calculus operations on polynomials that permit fractional and negative powers of variables. For example:

1. POLYCALC can expand an expression such as

$$(x + y^{1/2}/z)^2$$

into the equivalent generalized polynomial

$$x^2 + 2xy^{\frac{1}{2}z-1} + yz^{-2}$$

2. POLYCALC can determine that an indefinite integral of the above expression with respect to x is

$$x^{3/3} + x^2y^{\frac{1}{2}z-1} + xyz^{-2}$$

3. POLYCALC can determine that the partial derivative of the original expression with respect to y is

$$xy^{-\frac{1}{2}z-1} + z^{-2}$$

EXPRESSIONS

Variables

You can use any letter in the alphabet to represent a variable to which you assign a value. Thus, you can use as many as 26 such variables. POLYCALC interprets uppercase and lowercase letters to represent the same thing, so you may type in your variables either way.

Constants

Constants consist of adjacent digits, with an optional decimal point. For example: 82, 82., 8.2, .82 and 0.82.

Expressions

Expressions are formed largely following standard algebraic notation. However, POLYCALC does use slight modifications so you can enter data sequentially on one line. The conventions are as follows:

Arithmetic operators

Arithmetic operators appear between two subexpressions. Use + for indicating addition, - for subtraction, / for division, and * for multiplication. Use ^ to raise an expression to a power (called exponentiation). For example, X^2 represents X squared x^2 .

Multiplication operator

In contrast to most programming languages, POLYCALC multiplies adjacent subexpressions, whether or not you use an asterisk (*). For example, 5.2*37*X*4 and 5.2 37X4 both represent the product of four quantities, 5.2, 37, X, and 4.

You may use any number of blanks anywhere, except that you can't insert blanks within a constant. That is, POLYCALC interprets 32 differently than it interprets 3 2, but it interprets 3X the same as 3 X. However, you don't have to use blanks except between adjacent constants that the program would otherwise interpret as a single constant. For example, to multiply 5 by 6, type 5 6 or 5*6 rather than 56.

Scientific notation

You can use multiplication together with exponentiation to enter a number in scientific notation. For example, you can enter $3.8*10^{-6}$ as 3.8 10^-6. Since most integers this large aren't exactly representable in ATARI BASIC, the resulting display is 3.8E-6. In this context, ATARI BASIC uses the letter "E" to represent "*10^" where numerous zeros would otherwise be required to denote the number. However, you can't use this E-notation to enter problems in POLYCALC because it conflicts with the convention of implied multiplication and with using the letter E as a variable. POLYCALC output always precedes addition and subtraction operators by a space, so E-notation can be distinguished from all subexpressions containing the variable E. Nevertheless, you may prefer to avoid the letter E as a variable in order to avoid confusion.

```
After each question mark prompt, enter
an expression using digits, decimal
points, parentheses and 1-letter
variables together with operators
+, -, * and ^.  The latter means
raising to a power.  *, meaning
multiply, can be omitted. Each
equivalent expanded expression is
displayed if it and each subexpression
in the input can be represented as an
expanded polynomial generalized to
allow negative and fractional powers.
As examples, try

(X+Y)^2
(4A^6/B)^.5
1.5*10^2

After entering several such examples,
enter a question mark to display the
help topic menu.

?__
```

To try the practice problems, simply type one in and press the RETURN key. POLYCALC will display the results and you can verify your understanding of the lesson.

EXPRESSIONS

Variables

You can use any letter in the alphabet to represent a variable to which you assign a value. Thus, you can use as many as 26 such variables. POLYCALC interprets uppercase and lowercase letters to represent the same thing, so you may type in your variables either way.

Constants

Constants consist of adjacent digits, with an optional decimal point. For example: 82, 82., 8.2, .82 and 0.82.

Expressions

Expressions are formed largely following standard algebraic notation. However, POLYCALC does use slight modifications so you can enter data sequentially on one line. The conventions are as follows:

Arithmetic operators

Arithmetic operators appear between two subexpressions. Use + for indicating addition, - for subtraction, / for division, and * for multiplication. Use ^ to raise an expression to a power (called exponentiation). For example, X^2 represents X squared x^2 .

Multiplication operator

In contrast to most programming languages, POLYCALC multiplies adjacent subexpressions, whether or not you use an asterisk (*). For example, 5.2*37*X*4 and 5.2 37X4 both represent the product of four quantities, 5.2, 37, X, and 4.

You may use any number of blanks anywhere, except that you can't insert blanks within a constant. That is, POLYCALC interprets 32 differently than it interprets 3 2, but it interprets 3X the same as 3 X. However, you don't have to use blanks except between adjacent constants that the program would otherwise interpret as a single constant. For example, to multiply 5 by 6, type 5 6 or 5*6 rather than 56.

Scientific notation

You can use multiplication together with exponentiation to enter a number in scientific notation. For example, you can enter $3.8*10^{-6}$ as 3.8 10^-6. Since most integers this large aren't exactly representable in ATARI BASIC, the resulting display is 3.8E-6. In this context, ATARI BASIC uses the letter "E" to represent "*10^" where numerous zeros would otherwise be required to denote the number. However, you can't use this E-notation to enter problems in POLYCALC because it conflicts with the convention of implied multiplication and with using the letter E as a variable. POLYCALC output always precedes addition and subtraction operators by a space, so E-notation can be distinguished from all subexpressions containing the variable E. Nevertheless, you may prefer to avoid the letter E as a variable in order to avoid confusion.

Signed numbers

A + or - appearing to the left of a variable or constant represents its sign. For example, you enter a negative 3 as -3. Although such a + has no arithmetic effect, you can also enter it if you wish. For example, typing in 8/+--+-4, results in -2.

Order of operations

When no parentheses indicate otherwise, POLYCALC processes operators in this order:

```
first:  ^
        + and -   (when used as signs
                   preceding a number)
        * and /
last:   + and -   (when used between
                   subexpressions)
```

For example, POLYCALC evaluates the expression -2^2 as -(2^2), or -4, rather than as (-2)^2, or 4.

Multiple exponentiation

When no parentheses indicate other wise, POLYCALC computes successive exponentiations right to left. For example, 2^3^2 means 2^(3^2) which is 512, rather than (2^3)^2, or 8^2, which is 64.

Multiple multiplication and division

When no parentheses indicate otherwise, POLYCALC computes successive multiplications and divisions right to left. For example, 6/3*2 means (6/3)*2, or 4, rather than 6/(3*2), or 1.

Multiple addition and subtraction

When no parentheses indicate otherwise, POLYCALC performs successive additions and subtractions left to right. For example, 6-3+2 means (6-3)+2 rather than 6+(3-2), or 7.

Indicating division

In POLYCALC use the operator / to indicate division rather than building a fraction on more than one line. Therefore, you must use parentheses to get the desired effect when numerators or denominators have more than one term. For example, you enter

```
X + Y
-----
5 + 7
```

as (X+Y)/(5+7) but not as X+Y/5+7 or (X+Y)/5+7 or X+Y/(5+7).

Similarly, you must use parenthese for denominators of more than one factor unless you use repeated division to get the desired effect. For example, you can enter the fraction

$$\frac{N\ M}{D\ F}$$

as NM/(DF) or NM/D/F but not as NM/DF.

Integer exponents

You may use only positive whole numbers as exponents in POLYCALC if your base has more than one term. When your base has one term, you can use either positive or negative numbers as exponents.

IMPROPERLY FORMED EXPRESSIONS

Whenever POLYCALC encounters an improperly-formed expression, POLYCALC displays a ^ underneath the character at which it could no longer continue processing the expression left to right. Then, POLYCALC displays a brief error message indicating what kind of data it expected at that point. The program sometimes also reminds you of the availablility of the appropriate lessons. The message and location of the ^ often make the cause of the error obvious and you may not need to review the help topics or this manual. An example of faulty entry and POLYCALC's response is as follows:

```
? X + Y Z)^2
        ^

ERROR, EXPECTED ENTER, SEMICOLON,
QUESTION MARK, OPERATOR, DIGIT,
DECIMAL POINT, VARIABLE OR )
ENTER ANOTHER QUESTION MARK FOR HELP
```

It may be that the right parenthesis was intentional or that an earlier left parenthesis was accidentally omitted. Thus, the true "cause" of the error could be almost anywhere left of the indicated stopping point rather than at it. POLYCALC cannot read your mind to determine what you really meant. You'll need to edit or re-enter the expression, corrected as appropriate.

UNSTORABLE EXPRESSIONS

POLYCALC internally stores expressions as fully-expanded polynomials, generalized to permit fractional and/or negative powers of variables. Furthermore, the internal representation uses a particular order of the factors in a product and of the terms in a sum, which facilitates collection of similar factors and terms. For example:

$$(y+x)^2$$

is stored as

$$x^2 + 2xy + y^2$$

$$(4b^2/c)^{0.5}$$

is stored as

$$2bc^{-0.5}$$

$$(x+y)\ (y+1/x)$$

is stored as

$$xy + x^{-1}y + y^2 + 1.$$

Whenever POLYCALC encounters an expression that can't be represented internally, POLYCALC displays an appropriate error message following a ^ underneath the character at which it could no longer continue processing the expression left-to-right. For example, here are a few entry and response pairs:

```
 ? 2^X
     ^

ERROR, EXPECTED NUMERIC POWER BECAUSE BASE <> 1
ENTER ANOTHER QUESTION MARK FOR HELP

? 1/(X+1)
       ^

ERROR, EXPECTED 1-TERM DIVISOR
ENTER ANOTHER QUESTION MARK FOR HELP

(X+1)^0.5
       ^

ERROR, EXPECTED NONNEGATIVE INTEGER POWER BECAUSE
BASE HAS MORE THAN 1 TERM
ENTER ANOTHER QUESTION MARK FOR HELP
```

The restriction to nonnegative integer exponents of polynomials that don't simplify to one term is particularly important to note: POLYCALC is essentially a polynomial system rather than a rational expression or fractional power system. The ability to represent 1-term denominators and fractional powers of a single term is merely a useful conseqence of internally representing exponents of variables as "floating-point" numbers rather than as non-negative integers.

You must be able to represent the final result as a generalized polynomial as well as each subexpression of the entry. For example, although the product

$$(x+1)^{0.5}(x+1)^{1.5}$$

can be represented after collecting similar factors, POLYCALC does not continue processing the expression after discovering the fractional power of the first 2-term factor. Also, POLYCALC does not support general polynomial division. Therefore

```
? (X^2-1) / (X-1)
```

yields

```
ERROR, EXPECTED 1-TERM DIVISOR ENTER
ANOTHER QUESTION MARK FOR HELP
```

rather than

```
(X+1)
```

Accurracy and Rounding

To compute the coefficients and exponents of expanded results, POLYCALC uses the approximate floating-point arithmetic provided by ATARI BASIC. This arithmetic imposes a finite limitation on the precision of numbers. Thus, roundoff errors can accumulate with every step of the expansion process. Worse yet, the relative error in the difference of two nearly-equal numbers can be enormous compared to those of the two constituents. For example, suppose that we compute the difference.

```
 1.23456789
-1.23456666
-----------
 0.00000123
```

Even when you've computed the subtraction exactly, the difference has less significant digits than the least accurate operand. Thus, if accumulation of previous roundoff errors gave the least accurate operand only seven significant digits out of the nine that are stored, then the difference could have only one significant digit. Worse yet, if the least accurate operand has six or fewer accurate digits, then the result could have no accurate digits -- even the sign may be wrong. This cancellation phenomenon is a hazard with all systems that use approximate arithmetic -- not just ATARI BASIC.

With finite-precision in the representation of numbers, POLYCALC can store integers exactly, provided their size doesn't exceed the number of stored significant digits. ATARI BASIC's floating-point can also store fractions exactly if the numerators are not extreme and the denominators are small powers of 2. Thus, in order to postpone the introduction of roundoff errors as long as possible during the expansion process, it's best to phrase a problem using small magnitude integers and, either no fractions or, only ones having denominators that are sums of small powers of 2. For example, if the original problem is to expand

$$(100x + 200)^{10}$$

it is probably more accurate to expand the related expression

$$(x + 2)^{10}$$

Then mentally increase the resulting coefficients by a factor of $100^{10} = 10^{20}$

To provide further control over the effects of roundoff error, each computed coefficient

and exponent within a certain distance of an integer is rounded to that integer. No one distance can be appropriate for all problems, so this distance is initially zero. To change the distance, enter the desired new distance followed by the character "<". For example, to change the distance to 10^{-4}, enter

```
? 10^-4 <
CHANGED ROUNDOFF DISTANCE FROM 0 TO .0001
```

For problems that yield only integer coefficients and exponents, you may have to make the roundoff distance as large as 0.5. It's recommended that you use the initial roundoff distance of 0, increasing this as necessary to temporarily produce an exact result after studying the resulting roundoff errors. It's especially hazardous to use a roundoff distance greater than 0.1 without first studying a result produced by a smaller distance.

Limitations on magnitudes of numbers

Floating-point arithmetic also restricts the largest and smallest nonzero magnitudes of numbers. For ATARI BASIC, these magnitudes are respectively about 10^{38} and 10^{-38}

ATARI BASIC automatically produces 0 in place of an unrepresentably small-magnitude nonzero result. A factor or term attributable only to roundoff error might be destroyed by luck, but some legitimate terms might destroyed as well.

When a result has an unrepresentably large magnitude, POLYCALC displays the error message

```
ERROR, expected easier problem.
```

You can't do much about overflow except to work with less ambitious problems. However, sometimes you can reformulate a problem by changing a variable so as to involve lower degree and coefficient magnitudes closer to 1. Both actions reduce the likelihood of overflow.

ASSIGNMENTS

Be sure you have a good understanding of the concepts in the previous sections and have sufficient practice before tackling assignments.

Naming and saving a result for use in later expressions is often desirable for the following reasons:

1. Fit an entire expression in a single, one-line entry may be otherwise impossible.

2. Entering correctly a sequence of short formulas may be easier than entering an equivalent lengthly one.

3. Naming and saving a subexpression that occurs several places within an expression or that is to be used in several different expressions avoids repetitious entry.

POLYCALC uses the following form for assignments:

```
letter = expression
```

The effect of an assignment is to evaluate the expression then store the resulting value as that of the variable on the left side, replacing the former value of that variable. The resulting expanded value is displayed the same as if only the expression were entered.

The variables

Initially each of the 26 possible variables has its name as its value. Each expression is expanded immediately after you enter it. Whenever a variable is encountered in an input expression during this expansion process, the variable contributes the value that it has at that time. For POLYCALC, expansion an reduction combined with replacement of variables by their current values is called "evaluation".

Examples

Suppose we enter the assignment

```
? Y = X + 3
```

causing POLYCALC to display the new value of Y:

```
: X + 3
```

Next, suppose that we enter the assignment

```
? Z = Y^2
```

causing POLYCALC to display the new value of Z:

```
: X^2 + 6 X + 9
```

Now, here are two subtle points vital to understanding assignments. Suppose we enter the assignment

```
? Y = C + 5
```

causing POLYCALC to display the new stored value of Y:

```
: C + 5
```

If we then enter the expression

```
? Z
```

the resulting display is still

```
: X^2 + 6 X + 9
```

despite the changed value of Y and the fact that Z was assigned a value that is originally depended upon Y. The reason for this phenomenon is that after contributing the value X+3 to the expression Y^2, all traces of Y have disappeared from that result X^2+6X+9, which was assigned to Z. For efficiency, no record is kept of the fact that Z recieved its value from the evaluation of Y^2. Consequently, subsequent changes to Y

cannot have an effect on Z.

Now, suppose we enter the assignment

```
? X = W-1
```

causing the display

```
: W - 1.
```

If we then enter the expression

```
? Z
```

The resulting display is still

```
: X^2 + 6 X + 9
```

the same as before despite the changed value of X. X is explicitly present in the value that was assigned to Z, in contrast to the fact that we have no trace of Y. Consequently, this example is more surprising than the first one. However, whenever a variable receives a new value, POLYCALC does not automatically update other stored values that contain the name of the variable. This one-level POLYCALC evaluation mechanism is fast and simple to program. As well, one-level evaluation is sufficiently general to permit stepwise formation of any expression. We must simply enter the assignments in order "from the bottom" so that assignments of subexpressions to intermediate variables precede use of those variables in the final expression or in other intermediate expressions. For example, if the objective of the above session was to form Z in terms of W, then this objective could be accomplished by the sequence

```
? X = W-1
```

displaying

```
: W - 1
```

then entering

```
? Y = X+3
```

displaying

```
: W + 2
```

then entering

```
? Z = Y^2
```

displaying

```
: W^2 + 4 W + 4
```

Despite the use of an equal sign, regard a sequence of assignments as a sequence of updates to memory locations rather than as a set of simultaneous or independent equations.

POLYCALC allows you to assign a variable an expression containing the same variable. For example:

```
? X = X + 1
: X + 1

? X^2
:X^2 + 2X + 1
```

Although the result is consistent with the above discussion, the result may be surprising at first. Consequently, you may prefer to avoid the practice as much as possible by introducing additional variable names to achieve the intended effects. However, assignments containing the same variable on both sides are useful for increasing the numerical value of a variable as in the example

```
? X = 1
: 1
...
? X = X + 1
:2
...
?X = X + 1
: 3
...
```

The "..." represents computation involving X and perhaps other variables.

If there is no intent to use an algebraic or numeric result in subsequent expressions, then there is no reason to waste memory space by saving the result as the value of a variable. In such cases, it is easier and wiser merely to enter the desired expression.

Suppressed Display and Compound Entries

In sequences such as those of the last subsection, there are intermediate assignments for which the resulting display is of no interest. In such cases it may be desirable to suppress the displays in order to avoid their distraction and in order either to postpone scrolling off the top of a screen. Consequently, POLYCALC suppresses display of any expression or assignment followed by a semicolon.

For similar reasons, POLYCALC permits more than one expression and/or assignment per line, separated by semicolons. For example, the entry

```
? X=W-1; Y=X+3; Z=Y^2
```

results in the single display

```
: W^2 + 4 X + 4
```

Before proceeding to the next section, it is advisable to practice the concepts introduced up to this point.

Suppressed Evaluation

After assigning a value to a variable it is often desired to assign the variable its own name, which was the original value. This saves precious memory space and it clears the variable, permitting once again using it as an unencumbered name. For example, sometime after assigning X the value W-1, it may be desired to clear the value of X back to merely its own name. You might imagine that this could be accomplished with the assignment

```
? X = X
```

However, as mentioned before, all variables in an entry expression ordinarily are evaluated. Consequently, the right side of the above assignment would evaluate to W-1, which would be reassigned to X. What we want here is to suppress evaluation of a variable, and an apostrophe accomplishes this. If we enter

```
? X = 'X
```

then the displayed new value of X is the name X, as desired.

The "quote" operator is applicable only to variables, and the precedence of this operator exceeds all others.

Besides the above use, it is occasionally useful to apply a quote operator to variables within a more complicated expression in order to get the effect of clearing a variable locally within an expression without disturbing the assigned value outside the expression. For example, suppose that X has the value of W-1 that we wish to preserve, but that as a side calculation we wish to expand (X+1)^20 without having anything substituted for X. We could do so by merely entering the expression

```
? ('X + 1) ^ 20.
```

Re-evaluation

Suppose that for each of the values y=x+1, y=x+2, ..., y=x+10 we wish to evaluate the expression

$$2y^5 - y^4 + 7y^3 + 12y^2 - 13y + 863$$

We could do this in the bottom-up fashion as follows:

```
? Y=X+1; 2Y^5 - Y^4 + 7Y^3 + 12Y^2 - 13Y + 863
: 2X^5 + 9X^4 + 23X^3 + 47X^2 + 38X + 870

? Y=X+2; 2Y^5 - Y^4 + 7Y^3 + 12Y^2 - 13Y + 863
: 2X^5 + 19X^4 + 79X^3 + 190^2 + 247x + 989

                  . . .

? Y=X+10; 2Y^5 - Y^4 + 7Y^3 + 12Y^2 -13Y + 863
: 2X^5 + 99X^4 + 1967X^3 + 19622X^2 + 98327X + 198933
```

However, repetitious entry of the relatively lengthy fifth-degree polynomial is tiresome. This example is better suited to a top-down approach wherein we first assign the polynomial to another variable such as Z, then successively re-evaluate Z with Y assigned X+1, then X+2, etc.

POLYCALC provides the "@" for top-down stepwise evaluation of expressions. The precedence and left-to right order of this re-evaluation operator is the same as addition and subtraction. Consequently, to re-evaluate an entire expression, simply place "@" after the expression. For example, to do the above sequence of quintic examples top down, we could enter a sequence such as

```
? Z = 2Y^5 - Y^4 + 7Y^3 +12Y^2 - 13Y + 863;

? Y=X+1; Z@
: 2X^5 + 9X^4 + 79X^3 + 190X^2 + 247X + 989

              . . .

? Y=X+10; Z@
: 2X^5 + 99X^4 + 1967X^3 + 19622X^2 + 98327X + 198933
```

There is clearly much less total typing involved for this approach. Moreover, some user's often prefer to work in such a top-down manner: Starting with an overall "summary" value containing names, substitute subexpressions for some of the names in this value. Depending upon the particular expressions involved, the bottom-up, the top-down, or a mixed approach can involve less typing, memory consumption or computation time. Experiment with both extremes and various compromises to gain experience about which approach is most effective in various circumstances.

Note that the use of @ is somewhat different than that of the companion ALGICALC APX program.

It is important to note that each re-evaluation is only one level deep. For example, if the values of B, C and D are merely their own names, then the entered sequence

```
? A=B+1; B=C+2; C=D+5; A@
```

merely yields the display

```
: C + 3
```

whereas entering

```
? A @ @
```

yields

```
: D + 8
```

The ability to re-evaluate to any controllable level provides maximum flexibility. However, no re-evaluation is necessary in most common problems, and one re-evaluation is usually sufficient when re-evaluation is necessary.

It is also important to note that re-evaluating a variable does not automatically assign the new value to the variable. For example, if we continue the above sequence by entering the expression

```
? A
```

then the resulting display is

```
: B + 1
```

In contrast, if we continue with the entry

```
? A = A@@; A
```

then the resulting display is

```
: D + 8.
```

USING THE CALCULUS FEATURE

The following discussion assumes you're familiar with first-year calculus. You can ignore this section if you don't intend to use the calculus feature.

PARTIAL DERIVATIVES

The "%" denotes the partial derivative of its left operand with respect to the right operand. The character "%" is chosen to suggest a ratio of infinitesimals. The precedence and left-to-right direction of evaluation of this operator are the same as for addition. The right operand can be any expression that evaluates to a name. For example, if A and X have their own names as their values, then the entry

```
? AX^3 + X^-1.5 % X
```

results in the display

```
: 3 A X^2 - 1.5 X^-2.5
```

Higher-order derivatives can be specified by repeated use of the operator. For example, to add 5 onto the mixed second-order partial derivative of

$$x^3y^2$$

with respect to x and y, one could enter either

```
? X^3Y^2 % X % Y + 5
```

or

```
? 5 + (X^3Y^2 % X % Y)
```

Parentheses are necessary in the latter case because addition and differentiation have the same precedence and are done left to right.

Note that % is used slightly differently than in the companion ALGICALC APX program.

INTEGRALS

The "$" denotes the indefinite integral or antiderivative of its left operand with respect to the right operand. The character "$" is chosen to suggest the S-like integral sign. The precedence and left-to-right direction of evaluation of this operator are the same as for addition. The right operand can be any expression that evaluates to a name. For example, if A and X have their own names as their values, then the entry

```
? AX^3 + X^-1.5 $ X
```

results in the display

```
: 0.25 A X^4 - 2 X^-0.5
```

Note that as in most integral tables, a symbolic constant of integration is not included.

The indefinite integral of is , which is not representable in POLYCALC. Consequently, such logarithmic terms are announced with a warning that they are being discarded.

Re-evaluation can be used to compute a definite integral in cases where it can be computed by substitution into an indefinite integral that POLYCALC can accommodate. For example, the following dialogue computes

$$\int_2^Y cx^3\,dx:$$

```
? I = CX^3 $ X
: .25 C X^4

? X=Y; U=I@; X=2; L=I@; X='X; J=U-L
: -4C + .25CY^4
```

Moreover, this technique can be used to compute multiple definite integrals that can be done as iterated 1-dimensional integrals. For example, to compute

$$\int_0^1 \left(\int_2^Y cx^3\,dx\right) dy$$

we can start as above, then continue with:

```
? Y=1; U=J@; Y=0; L=J@; Y='Y; U-L
: 3.95C
```

TRUNCATED POWER SERIES

Terms containing a variable raised to a power that exceeds a certain threshold are replaced by zero at every opportunity during evaluation. This threshold is initially 10^{36} so that terms are not ordinarily discarded for this reason. However, you can change the threshold to a much smaller value, which permits obtaining the low-degree terms of an expansion that would otherwise exceed the memory capacity of the coefficient magnitude limit or your patience. To change the degree threshold simply enter the desired new numeric value followed by the character ">". This also causes the previous value of threshold to be displayed, making it easier to remember an appropriate value to restore it to later. You may wish to jot it down for future restoration.

For example, in probability theory it is often desired to determine the number of distinct subsets of m items drawn from a set of n items. It can be shown that the number is the coefficient of x^m in the expansion of $(x+1)^n$. As a specific application, how many distinct five-card poker hands are there in a deck of 52 unique cards? Rather than compute all 53 terms of $(x+1)^{52}$ we can determine the coefficient of x^5 much faster by limiting all intermediate and final powers to a maximum of 5 as follows:

```
? 5 >
CHANGED MAXIMUM DEGREE FROM 1E+36 TO 5

? (X+1) ^ 52
2598960X^5 + 270725X^4 + 22100X^3 +1326X^2 + 52X + 1
```

Thus, there are 2,598,960 distinct 5-card poker hands, 270,725 distinct 4-card poker hands, and so on.

As the next example, suppose that we wish to determine the fourth-degree truncated power series expansion of $e^{\cos x}$ expanded about the point x=0. Manually using the general Taylor-series formula or referring to math tables, we begin with the component fourth-degree expansions

$$\cos x = - x^{2/2} + x^{4/24}$$
$$e^{c} = 1 + c + c^{2/2} + c^{3/6} + c^{4/24}$$

We can compose these two in a bottom-up fashion as follows:

```
? 4 >
CHANGED MAXIMUM DEGREE FROM 5 TO 4

? C = 1 - X^2/2 + X^4/24; 1 + C + C^2/2 + C^3/6 + C^4/24
: .423611111X^4 - 1.33333333X^2 + 2.70833334
```

Alternatively, we could compose the two series in a top-down fashion as follows:

```
? 4 >
CHANGED MAXIMUM DEGREE FROM 5 TO 4

? C='C; E=1+C+C^2/2+C^3/6+C^4/24; C=1-X^2/2+X^4/24;
E@
: .423611111X^4 - 1.33333333X^2 + 2.70833334
```

Beware that cancellations of factors can necessitate performing intermediate expansions to a higher degree than the desired result. For example, suppose that we want a third-degree truncated expansion of $\frac{e^x - 1}{x} + \sin x$

Starting with the two third degree truncated Taylor series expansions

$$e^x = 1 + x + x^{2/2} + x^{3/6} + \ldots$$
$$\sin x = x - x^3/6 + \ldots,$$

if we enter

```
? 3 >
CHANGED MAXIMUM DEGREE FROM 4 TO 3

? ((1 + X + X^2/2 + X^3/6) - 1) / X + (X - X^3/6)
```

then resulting display is

```
: -.166666667X^3 + .166666667X^2 + 1.5X + 1
```

which is valid only through degree 2 even though a third-degree term appears -- we should have used a fourth-degree series for e^x

Another precaution is to beware that the same degree threshold applies to each variable,

as if all variables are equally important in deciding whether or not to retain a term. When this treatment is inappropriate, then the desired effect can be obtained by a change of variables such that the same degree threshold for all of the new variables corresponds to the various desired thresholds for the original variables. For example, if it is desired to have a degree threshold of 2 for x, 3 for y and infinity for c in the expansion

$$(c^7 + x + y)^5$$

then we can replace x by U^3, y by V^2, and c by K^(1/7), then proceed as follows:

```
? 6 >
CHANGED MAXIMUM DEGREE FROM 3 TO 6

? P = (K + U^3 + V^2)^5
: K^5 + 5K^4U^3+5K^4V^2 + 10K^3U^6+ 20K^3U^3V^2
+ 10K^3V^4 + 30K^2U^6V^2 + 30K^2U^3V^4 + 10K^2V^6
+30KU^6V^4 + 20KU^3V^6 + 10U^6V^6
```

If desired, we can further proceed to restore the original threshold of 10^{36}, then use re-evaluation to express the result in terms of the original variables:

```
? 10^36 >
CHANGED MAXIMUM DEGREE FROM 6 TO 1E+36

? K=C^7; U=X^(1/3); V=Y^.5; P@
:C^35 + 5C^28X + 5C^28Y + 10C^21X^2 + 20C^21XY
+30C^7X^2XY^2 + 20C^7XY^3 + 10X^2Y^3
```

As a final note, for some series calculations it may be useful to use a nonzero roundoff distance as well as a maximum degree to discard terms.

HOW TO LEARN MORE ABOUT COMPUTER ALGEBRA

We expect that many who experience POLYCALC or another computer algebra system will want to learn more about this fascinating subject or will want to try other systems that have complementary or more powerful capabilities. Accordingly, here is a brief guide to the relevant professional societies, the literature, and some widely available systems.

The Professional Societies

The Association for Computing Machinery Special Interest Group on Symbolic and Algebraic Manipulation is the major international professional society for computer algebra. Their ACM SIGSAM Bulletin is the most concentrated and up-to-date source for abstracts and working papers together with announcements of meetings and systems. For information about joining, including special student rates, write the ACM at 1133 Avenue of the Americas, New York, NY 10036.

Some European groups devoted to computer algebra are:

1. SAM-AFECT: Contact M. Bergman, Faculte des Sciences de Luminy, Case 901, 13009; or contact J. Calmet, IMAG, B.P. 53, 38041 Grenoble Cedex, France.

2. NIGSAM: Contact Y. Sundblad, Department of Numerical Analysis and Computer Science, KTH S-10044 Stockholm, Sweden.

3. SEAS/SMC: Contact J. A. van Hulzen, Twente University of Technology, P.O. Box 217, 7500 AE enschede, The Netherlands.

The Literature

Regrettably, there is not yet a textbook devoted to computer algebra, and the few textbooks that contain relevant material are rather advanced. Most of the information is sparsely scattered in research journals or less accessible conference proceedings and reports. However, the following relatively accessible references contain surveys, bibliographies, and collections of articles that should serve as a good point of departure for exploring most facets of the literature:

1. ACM SIGSAM Bulletin, ACM, New York, all issues.

2. Communications of the ACM 14, No. 10, August 1971.

3. SIAM Journal on Computing 8, No. 3, August 1979.

4. Communications of the ACM 9, No. 10, August 1966.

5. Journal of the ACM 18, No. 4, October 1971.

6. P.S. Wang, editor, proceedings of the 1981 ACM Symposium on Symbolic and Algebraic Computation, ACM Order No. 505810, P.O. Box 64145, Baltimore, MD 21264, $23.

7. E.W. Ng, editor, Symbolic and Algebraic Computation, Lecture Notes in Computer Science, 72, Springer-Verlag, New York, 1979.

8. R.D. Jenks, editor, Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computation, ACM, New York, 1976.

9. V.E. Lewis, editor, Proceedings of the 1979 MACSYMA User's Conference, M.I.T. Laboratory for Computer Science, 545 Technology Square, Cambridge, Massachusetts, 1979.

10. C.M. Anderson, editor, Proceedings of the 1977 MACSYMA User's Conference, NASA CP-2012, 1977.

11. Knuth, D.E., The Art of Computer Programming, Volume II, Seminumerical Algorithms, Addison-Wesley, Reading, Mass., 1980.

12. Yun, D.Y.Y and Stoutemyer, D.R., "Symbolic Mathematical Computation", Encyclopedia of Computer Science and Technology, Supplementary Volume 15, J. Belzer, A.G. Holzman and A. Kent, editors, M. Dekker, New York, pp. 235-310.

Other Computer Algebra

POLYCALC has an analogous companion program that is about the same size and is distributed predominately for the same machines by the same software publishers: ALGICALC treats expressions that can simplify to ratios of polynomials in the unassigned variable X, which may be assigned to any other variable A through Z. Thus in comparison to POLYCALC, the class of allowable expressions is broader in some respects and narrower in others. Expansions, factoring, differentiation and numeric substitution are supported, so the allowable operations are also broader in some respects and narrower in others. The net result is that together the two programs span a significantly larger range of problems than either package alone.

Besides POLYCALC and ALGICALC, there are also larger systems that provide more capabilities at the expense of greater demands on memory space, computer sophistication and cost. In contrast to POLYCALC and ALGICALC:

1. They require anywhere up to 200 times as much memory space.

2. They accommodate a significantly larger class of expressions, including general rational or algebraic expressions in any number of variables, perhaps with logarithmic, exponential and trigonometric extensions. Some of these systems can even accommodate equations, matrix or tensor expressions directly.

3. They provide their own indefinite-precision arithmetic to avoid the serious limitations of finite-precision arithmetic.

4. They accommodate much larger expressions having thousands of terms, with coefficients having hundreds of digits.

5. They provide a larger suite of built-in and optional transformations, including factoring and partial-fraction expansions for example.

6. They provide convenient facilities permitting the user to write function definitions and simplification rules to extend the built-in capabilities by enlarging the allowable class of expressions or the variety of available transformations. In other words, they are programmable.

In appropriate order of increasing memory requirements, here are some of the most widely available general-purpose systems that are currently supported:

1. muMATH-80tm is an interactive system that runs on personal microcomputers based on the 6502, 8080, 8085, or Z80 microprocessors, provided they have enough memory and an appropriate disk operating system. These include CP/M with at least 32 kilobytes of RAM memory, the Radio-Shack TRS-DOS with at least 32 kilobytes of such memory, or the Apple computer with at least 48 kilobytes of such memory. muMATH is ditributed to end users, computer stores and hardware manufactures by Microsoft at 10800 N.E. Eighth, Suite 819, Bellevue Washington 98004, and muMATH is also distributed by the authors, The Soft Warehouse, at Box 11174, Honolulu, Hawaii 96828. The less common CP/M disk formats are available from Lifeboat Associates, 1651 Third Avenue, New York, N.Y. 10028.

2. SAC-2 is a non-interactive system which runs on any computer that can directly run a 1966 standard FORTRAN program of at least about 120 kilobytes. Information about SAC-2 is available from Professor George Collins, Computer Sciences Department, University of Wisconsin, 1210 West Dayton St., Madison, Wisconsin 53706.

3. FORMAC runs on any IBM 360 or 370 that can accommodate a PL/I program of at least about 150 kilobytes . FORMAC is semi-interactive on some operating systems. Information about FORMAC is available from Knut Bahr at GMD/IFV, D-6100, Darmstadt, Germany.

4. ALTRAN is a non-interactive system which runs on any computer that can directly run a 1966 standard FORTRAN program of at least about 270 kilobytes. Information about ALTRAN is available from the Computing Information Library, Bell Laboratories, 600 Mountain Avenue, Murray Hill, N.J. 07974.

5. REDUCE is an interacitve system that runs on the IBM 360 or 370, DEC 10 or 20, Univac 1100 series, Control Data Cyber series, Burroughs 6700, and several other computers, requiring a minimum of about 350 kilobytes. For information about REDUCE, write Dr. Anthony Hearn, Rand Corporation, 1700 Main Street, Santa Monica, California 90401.

Additional systems are announced in back issues of the ACM SIGSAM Bulletin.

COMPUTER ALGEBRA IN EDUCATION

It should be clear to anyone who has experienced a computer-algebra system that it has enormous potential for use in education as well as research. Not only can computer algebra make computing more attractive to mathematically inclined students; computer algebra can make mathematics more attractive to computer enthusiasts. Thus, computer algebra provides a great opportunity for mutal reinforcement and cross motivation between math and computer education.

Personal computers are becoming so prevalent that students, engineers, scientists, and mathematicians will soon be using computer algebra extensively. Moreover, it should not be long before general-purpose computer algebra is available on pocket calculators, because:

1. Several manufactures now make low wattage "CMOS" versions of most popular 8-bit microprocessors, and CMOS versions of some 16-bit microprocessors are currently under development.

2. There are already hand-held terminals with 32 kilobytes of low wattage reak-write memory.

3. There are already hand-held calculators with a sufficiently large low wattage liquid-crystals to display a reasonably large mathematical expression -- perhaps one term at a time.

Eventually some enterprising manufacturer will surely merge these three technologies, producing a hand-held calculator capable of running a general-purpose computer-algebra system such as mumMATH. Thus, it behooves every math and computer science educator to explore how this revolutionary tool can be used to aid education.

It is undeniably true that most students are far more intrigued and motivated by the artificial intelligence and game playing applications of computers than by the accounting and numerical applications that currently account for most computer usage. Thus, it is advisable to exploit this strong preferential interest to help teach both mathematics and computer science. If more good math, science, and engineering students are attracted to computers and more good computer-oriented students are attracted to math, then more students will ultimately learn to use computers effectively for both numeric and nonnumeric purposes.

Computer algebra makes a highly motivating introductory computer programming course for math, science and engineering students. Computer algebra is also an ideal principal language for such students, because numbers and arithmetic comprise appreciably less than half of the kindergarden through calculus math curriculum. Moreover, the limited-precision integer and floating-point arithmetic typical of traditional programming languages is not the kind of arithmetic taught in this curriculum or used in everyday life.

Some educators may fear that computer algebra might cause algebraic skills to artophy or prevent them from ever delveloping. Analogous concerns were undoubtedly expressed about Arabic numerals, multiplication tables, logrithms, Laplace transforms and numerical

pocket calculators; but we have survived their convenience. The National Council for Teachers of Mathematics strongly supports the use of numerical pocket calculators in classrooms, and every reason for this support is even more true of computer algebra.

Automatic symbolic mathematics makes it possible for students to concentrate on basic mathematical concepts rather than spending an excessive amount of time mechanically performing transformations. Computer algebra lets students explore such fundamental concepts as commutativity, associativity, groups and rings. Moreover, the extensive algebraic capabilities of computer algebra enables students to investigate larger examples than is otherwise practical. Patterns thus revealed may suggest useful theorems. Conjectured patterns thus violated provide counterexamples against false hypotheses. Thus, computer algebra can contribute to teaching mathematical discovery.

Existing computer-algebra systems could also make other educational contributions:

1. Trace packages can be used to let students witness each step of algebraic simplification, rather than merely the final result.

2. The very fact that algebra and calculus can be automated should encourage average and poor math students that the flashes of inspiration characteristic of quick students are unnecessary for those operations -- there is revealed hope for the slower more methodical students.

3. For students who know how to program in the language in which the computer algebra packages are written, inspection of the underlying algorithms may help them learn methods for accomplishing the operations. Moreover, by programming extensions to the built-in facilities, students can reinforce understanding of the built-in and extended operations.

More existing computer algebra systems can be used in the above ways right now. However, there is a potential for much more. In conjunction with a computer-aided instruction package, existing computer algebra systems could be used for extremely flexible and intelligent algebra drill, testing, and tutorial dialogue.

None of the existing computer algebra systems is by itself a computer-aided math instruction system. However, someone experienced in computer-aided instruction could design interactive math lessons or examinations to be used in conjunction with these existing computer algebra systems. ould design interactive math lessons or examinations to be used in conjunction with these existi

ALPHABETIZED ERROR MESSAGES: PROBABLE CAUSES

The POLYCALC error messages are detailed and precise, indicating exactly where processing could not continue and why. For example, if the error was caused by an ill-formed expression, then POLYCALC states every category of character that is allowable at that point. This section lists all of these messages in alphabetical order, along with some of the most common causes.

Expected +, -, /, semicolon, digit, decimal point, variable or (:

Perhaps an operator or a parenthesis occurs where it is not allowed, such as in the entries "X**2", ")", "()", or "*". Alternatively, perhaps there is an illegal character such as "!", "#" or "]".

Expected), operator, digit, decimal point, variable or (:

POLYCALC encountered the end of an expression while seeking a right parenthesis to match an earlier left parenthesis.

Expected 1-term divisor:

POLYCALC cannot divide by factors that evaluate to more than one term.

Expected a nonzero divisor:

An attempt was made to divide by a factor that evaluates to zero, and division by zero is mathematically undefined.

Expected a term whose value is a variable: see lessons 11 & 12:

An attempt was made to differentiate or integrate with respect to a term that does not evaluate to the name of a variable. Perhaps the differentiation or integration variable has an assigned value that should be cleared by quoting.

Expected enter, question mark, semicolon, operator, digit, decimal point, variable or (:

There are extra characters that cannot be incorporated in the indicated complete expression. Common causes are excess right parentheses or illegal characters such as "!", "#", or "]".

Expected existing lesson number:

The expression preceding a question mark must evaluate to one of the integer lesson numbers listed in the lesson menu.

Expected integer power because base is negative or is one term with a negative coefficient:

POLYCALC cannot represent the imaginary numbers that correspond to fractional powers of negative numbers or coefficients.

Expected nonnegative integer power because base is sum:

POLYCALC cannot treat negative or fractional powers of an expression to more than one term.

Expected nonnegative power because base = 0:

Zero raised to a negative power is equivalent to division by zero, which is mathematically undefined.

Expected nonnegative roundoff distance: see lesson 4:

The expression preceding "<" must evaluate to a nonnegative numeric distance below which numbers are rounded to integers.

Expected numeric maximum degree: see lesson 10:

The expression preceding ">" must evaluate to a numeric degree beyond which terms are discarded.

Expected numeric power because base <> 1:

1 raised to any power is 1, but POLYCALC cannot otherwise treat exponential subexpressions, such as 2^X or 2.71828^X.

Warning, discarded result term ... see lesson 12:

POLYCALC cannot represent the natural logarithm that results from intergrating a term containing the reciprocal of the integration variable.

Any other error messages are generated by ATARI BASIC or DOS in a manner that was unanticipated by The Soft Warehouse. See your Atari Reference Manual.

QUICK REFERENCE SHEET
FOR POLYCALC

To enter problems --> enter data [RETURN]
To request HELP menu --> ? [RETURN]
To request a lesson --> lesson number ? [RETURN]

VARIABLES --> A/a - Z/z

ORDER OF OPERATIONS:

^, - (as signed numbers)
*, / (multiplication, division)
+,-,@,%,$ (addition, subtraction, reevaluation, differentiation, integration)

MULTIPLE EXPONENTIATION: right to left, when no parentheses
MULTIPLE MULTIPLICATION AND DIVISION: left to right, when no parentheses
MULTIPLE ADDITION, SUBTRACTION, SUBSTITUTION, DIFFERNTIATION: left to right when no parentheses

Use PARENTHESES IN DIVISION to include more than one factor (e.g., xy) as a denominator

EXPONENTS --> for nonnegative integers only, if base is a sum

Arithmetic limitations of ATARI BASIC
Magnitude limitation for integers to ,avoid roundoff errors:
LARGEST NONZERO MAGNITUDE: 10^96
SMALLEST NONZERO MAGNITUDE: 10^-98

ASSIGNMENTS: letter = expression

To clear a variable use: variable= 'variable

SUPPRESSED DISPLAY --> end input with ;

To enter MORE THAN ONE EXPRESSION/ASSIGNMENT PER LINE --> separate them with ;

SUBSTITUTION of numerical value for X: (sub)expresseion @ numberic value

PARTIAL DERIVATIVES of expressions or subexpressions with respect to a variable:
(sub)expression % variable
Repeat % to express higher order derivatives

ANTIDERIVATIVES of expressions or subexpressions with respect to a variable:
(sub)expression $ variable

To CHANGE roundoff threshold enter:

new threshold<

To CHANGE power truncation threshold, enter:
new threshold >

LIMITED WARRANTY ON MEDIA AND HARDWARE ACCESSORIES.

We, Atari, Inc., guarantee to you, the original retail purchaser, that the medium on which the APX program is recorded and any hardware accessories sold by APX are free from defects for thirty days from the date of purchase. Any applicable implied warranties, including warranties of merchantability and fitness for a particular purpose, are also limited to thirty days from the date of purchase. Some states don't allow limitations on a warranty's period, so this limitation might not apply to you. If you discover such a defect within the thirty-day period, call APX for a Return Authorization Number, and then return the product along with proof of purchase date to APX. We will repair or replace the product at our option.

You void this warranty if the APX product: (1) has been misused or shows signs of excessive wear; (2) has been damaged by use with non-ATARI products; or (3) has been serviced or modified by anyone other than an Authorized ATARI Service Center. Incidental and consequential damages are not covered by this warranty or by any implied warranty. Some states don't allow exclusion of incidental or consequential damages, so this exclusion might not apply to you.

DISCLAIMER OF WARRANTY AND LIABILITY ON COMPUTER PROGRAMS.

Most APX programs have been written by people not employed by Atari, Inc. The programs we select for APX offer something of value that we want to make available to ATARI Home Computer owners. To offer these programs to the widest number of people economically, we don't put APX products through rigorous testing. Therefore, APX produts are sold "as is", and we do not guarantee them in any way. In particular, we make no warranty, express or implied, including warranties of merchantability and fitness for a particular purpose. We are not liable for any losses or damages of any kind that result from use of an APX product.

APX ATARI® PROGRAM EXCHANGE
P.O. Box 3705
Santa Clara, CA 95055

Review Form

We're interested in your experiences with APX programs and documentation, both favorable and unfavorable. Many of our authors are eager to improve their programs if they know what you want. And, of course, we want to know about any bugs that slipped by us, so that the author can fix them. We also want to know whether our instructions are meeting your needs. You are our best source for suggesting improvements! Please help us by taking a moment to fill in this review sheet. Fold the sheet in thirds and seal it so that the address on the bottom of the back becomes the envelope front. Thank you for helping us!

1. Name and APX number of program.

2. If you have problems using the program, please describe them here.

3. What do you especially like about this program?

4. What do you think the program's weaknesses are?

5. How can the catalog description be more accurate or comprehensive?

6. On a scale of 1 to 10, 1 being "poor" and 10 being "excellent", please rate the following aspects of this program:

   _____ Easy to use
   _____ User-oriented (e.g., menus, prompts, clear language)
   _____ Enjoyable
   _____ Self-instructive
   _____ Useful (non-game programs)
   _____ Imaginative graphics and sound

7. Describe any technical errors you found in the user instructions (please give page numbers).

8. What did you especially like about the user instructions?

9. What revisions or additions would improve these instructions?

10. On a scale of 1 to 10, 1 representing "poor" and 10 representing "excellent", how would you rate the user instructions and why?

11. Other comments about the program or user instructions:

From

STAMP

ATARI Program Exchange
P.O. Box 3705
Santa Clara, CA 95055

[seal here]